Choreographing Web Services with Semantically Enhanced Scripting

Bai, X., Klein, E., & Robertson, D. (2012). Choreographing Web Services with Semantically Enhanced Scripting. In Web Intelligence and Intelligent Agent Technology (WI-IAT). (pp. 583-587). IEEE. 10.1109/WI-IAT.2012.127

Choreographing Web Services with Semantically Enhanced Scripting

Xi Bai*†, Ewan Klein*, Dave Robertson*
*School of Informatics, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB, UK
†Microart, Parc Cientfic de Barcelona
Barcelona 08028, Spain
Email: xi.bai@ed.ac.uk, {ewan,dr}@inf.ed.ac.uk

***Abstract*—Several solutions to describing service choreography have emerged, mainly focused on encoding capabilities of services especially for those deployed on the Web. These solutions are either derived from traditional Web service standards such as WSDL or inspired by the theory of process calculus. Little attention has however been paid to finding a lightweight solution which can enable peers to obtain, publish and share service choreography in an open environment or peer-to-peer network. This paper proposes a framework for choreographing semantically enhanced Web Services encoded in a extended lightweight coordinative language which is derived from *process calculus* and is dedicated to running in modern Web browsers. A proof-of-concept prototype has been implemented and demoed as a decentralised service-choreography-management platform based on this framework. There is no need for users to install any third-party application, and service choreography execution is achieved via client-side Web browsers. Also, the preliminary experiments indicate the efficiency and scalability of our proof-of-concept implementation of this framework.**

***Keywords*-web service choreography; linked data; process calculus;**

I. Introduction

Web Services (WSs) have received considerable attention within both academia and industry as a means of virtualising software and thereby building scalable decentralised systems on the Internet. We will focus in this paper on peer-based WS *choreography*, which we interpret as a top-down perspective on WS coordination where all services participate as equals but interact in conformance to a specification of social norms in the peer-to-peer networks. This contrasts with *orchestration*, which focuses on the behaviour of a single service coordinating the interaction (with other services only being involved as required by the orchestrator). Several vocabularies inspired by WSDL have been proposed for semantically enhancing WS descriptions but comparatively little attention has been paid to the semantic aspects of WS choreography running in an open *ad hoc* environment or peer-to-peer networks. To address this, we will focus on a minimal language with just these concepts (e.g., constraints and their interaction with message passing, etc.), taking as our starting point the notion of the Interaction Model (IM) encoded in the Lightweight Coordination Calculus (LCC) [1].[1] LCC is a declarative language used by OpenKnowledge for describing choreography and employed in this paper due to its lightweight expression and executability.

An IM is a set of clauses defining the behaviours associated with roles within peer interactions.[2] A *role* describes the necessary actions for each of the peers taking part in the interaction. We show in this paper how to extend LCC to a new choreography description language, XLCC, which remains compact enough to execute as a script on various devices installed with modern browsers. The syntax of XLCC is described in BNF in Figure 1.

IM	:=	$Clause\_List$
$Clause\_List$	:=	$Clause \mid Clause\_List$
$Clause$	:=	$Role :: Def. \mid Role. \mid plays(Constant,\ Constant). \mid knows(Constant). \mid iid(Constant).$
$Role$	:=	$a(Type,\ Id)$
Def	:=	$Message \mid Def\ then\ Def \mid Def\ or\ Def \mid Def\ niob\ Def$
$Message$	:=	$M \Rightarrow Role \mid M \Rightarrow Role \leftarrow C \mid M \Leftarrow Role \mid C \leftarrow M \Leftarrow Role \mid null \leftarrow C \mid Role \mid Role \leftarrow C$
C	:=	$Constant \mid Constant(Terms) \mid not(C) \mid C \,\&\&\, C \mid C \mid\mid C \mid list(Variable,\ Variable,\ Variable)$
$Terms$	:=	$Term,\ Terms \mid Term$
$Type$	:=	$Term$
Id	:=	$Constant \mid Variable$
M	:=	$Constant(Term)$
$Term$	:=	$Constant \mid Variable \mid Constant(Terms) \mid \_$
$Constant$	:=	a string starting with a lower case character
$Variable$	:=	a string starting with an upper case character

Figure 1. XLCC syntax

After being encoded in XLCC, IMs can be annotated with the WSCAIM (Web Service Choreography As Interaction Models) vocabulary, which can comprehensively describe IM-driven WS choreography and benefit service discovery/repurposing. The OKeilidh system[3] is an online decentralised platform built on top of Web browsers and allows peers to publish, annotate and execute WS choreography modelled as IMs via the components illustrated in Figure 2. By embedding metadata in (X)HTML, publishers can attach semantics to Web content, which makes the Web page itself both machine-readable and human-readable. Sev-

[1] Cf. also the OpenKnowledge system http://www.openk.org

[2] We follow the example of the OpenKnowledge system in using the term *peer* (rather than *agent*) to focus on reactive behaviours of participants within interaction.

[3] http://www.openk.org/okeilidh/

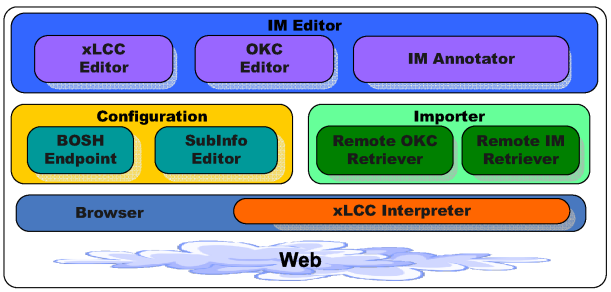

Figure 2. Architecture overview

eral solutions to embedding metadata into Web pages have been proposed, including Microformats [2], RDFa[3] and Microdata[4]. In this paper, we will restrict our attention to RDFa, which is superior to other solutions by not only taking advantage of standard XHTML markup but also providing several new XHTML attributes for achieving flexibility and disambiguation. RDFa also reuses the existing RDF model and supports full RDF semantics.

The remainder of this paper is organised as follows. Section II introduces XLCC designed as an extension of LCC and elaborates a new asynchrony operator. Section III proposes a new vocabulary for semantically enhancing the WS choreography specification shared by peers. The evaluation of a new asynchrony operator is presented in Section IV. Section V gives a brief description on the work related to this paper. The conclusion is drawn in Section VI and the direction of future work is also indicated.

II. Overview of xLCC

The XLCC language was derived from LCC, which is a compact declarative language [5] deployed in the OpenKnowledge project for describing and executing WS choreography between peers. LCC choreography specifications are also executable, but until now LCC interpreters have only been produced for peer-to-peer and service architectures, not for a Web environment. Moreover, LCC still faces concurrency-design issues which have not been tackled in the OpenKnowledge kernel. By contrast, the XLCC script allows publishers to create IMs which can be executed in modern browsers. Normally, service orchestration systems operate by installing server applications or, in the case of choreography, downloading an application on every peer. However, it is possible to obtain the same functionality through a Web browser, which frees users from the need to download applications or reconfigure port settings. We demonstrate this for XLCC, although the same principle could apply to other process languages executable in browsers. As shown in Figure 2, the XLCC script and the *XLCC Interpreter* component are focused on IMs published via Web documents (in (X)HTML) and corresponding run time environments (e.g., browsers). Figure 3 describes a journey-planning IM in xLCC and it involves six roles including one role-change (due to the limited paper space, several role bodies are omitted and substituted by $\sim$). First, a traveller T sends to a travel agent TA the times and locations of her departure and arrival. Second, TA normalises the query and sends it to a CRS (Carrier Routing System) C which will generate routes from the journey start and endpoint information. TA also sends the journey query to an evaluation unit E in order to constantly get latest statistics on travellers' queries. Third, C sends each generated route to a GDS (Global Distribution System) G to obtain costs for each route and then sends journey information back to TA, which will reprice each journey and also generate final options for T. After receiving a message with journey options from TA, T makes a choice and notifies a TMC (Travel Management Company) TM for booking by her credit card. Finally, TM sends the ticket and the receipt back to the interaction initiator T.

```
a(traveller, T)::
  search(Departure, Arrival, DepTime, ArrTime)
    ⇒ a(tavelAgent, TA) then
  display(Options) ← options(Options) ⇐ a(travelAgent, TA)
   then
  book(JourneyID, CC) ⇒ a(tmc, TM)
    ← chooseJourney(Options, JourneyID) && payby(CC) then
  booked(Tickets, Receipts) ⇐ a(tmc, TM).

a(travelAgent, TA)::
  search(Departure, Arrival, DepTime, ArrTime)
    ⇐ a(traveller, T) then
  journeyQuery(Query) ⇒ a(crs, C)
    ← normalise(Departure, Arrival, DepTime, ArrTime, Query)
   then
  recommend(Journeys) ⇐ a(crs(Routes, Journeys), C) then
  options(Options) ⇒ a(traveller, T)
    ← repricing(Journeys, Options) then -----> niob
  record(Query) ⇒ a(evaluator, E) then
  evaluation(Statistics) ⇐ a(evaluator, E).

a(crs, C)::∼. a(crs(Routes, Journeys), C)::∼.

a(gds, G)::∼. a(evaluator, E)::∼.

a(tmc, TM)::
  book(JourneyID, CC) ⇐ a(traveller, T) then
  booked(Tickets, Receipts) ⇒ a(traveller, T)
    ← charge(JourneyID, CC, Receipt) && print(Ticket).
```

Figure 3. Basic Travel Planning IM in XLCC

A. Design of the Asynchrony Operator

An IM describes the interaction of peers and their associated obligations. The latter are encoded as constraints whose solvers are wrapped into so-called OpenKnowledge Components (OKCs), which can be retrieved from external repositories or created locally on the fly (cf. Figure 2). Every IM models a process driven by diverse events, including constraint solving and message passing, and significant overhead during the IM execution is caused by message passing. For instance, in Figure 3, the travel agent T constantly receives updated statistics on the journey query history in order to find out travellers' preferences and improve its own recommendation system. This interaction is orthogonal to other interactions arising from booking a journey. It will however

be unnecessarily blocked by the CRS C if C cannot send the message *recommend(Journeys)* back to T in time. This inefficiency is caused by use of the operator `then` (highlighted in the left box in Figure 3), which requires blocked I/O—an input/output-processing pattern that prohibits other processing until the transmission has finished). Although message-intensive choreography demands the careful design, so far little attention has been paid to the potential scalability issue.

By taking advantage of both thread-driven programming and event-driven programming, we propose an efficient method of interpreting IMs that incorporates a non-blocking I/O mechanism. Specifically, we add to XLCC an asynchrony operator `niob`, interpreted via the event-based asynchronous design pattern. Modern browsers normally are single-thread-ed (Google Chrome has one process for each tab in a single window) and support non-blocking I/O natively. Our XLCC script interpreter has been designed and implemented to run in widely adopted browsers.

Standard LCC uses two operators to sequence message passing in a clause: `then`, as mentioned earlier, and `or`. S_1 `then` S_2 requires S_1 to be completed first, after which S_2 will be completed; S_1 `or` S_2 stipulates that either S_1 or S_2 will be completed and that S_1 will be attempted first. After adding the new operator `niob`, we adopt the following interpretations (in JavaScript callback functions) of the three sequence operators:

Table I
INTERPRETING SEQUENCE OPERATORS IN XLCC

S_1 `then` S_2	`execute(`S_1`, function(satisfied) {`
	`  if (satisfied) execute(`S_2`);`
	`});`

S_1 `or` S_2	`execute(`S_1`, function(satisfied) {`
	`  if (!satisfied) execute(`S_2`);`
	`});`

S_1 `niob` S_2	`execute(`S_1`); execute(`S_2`);`

As shown in Table I, callback functions are used for ensuring a strict execution sequence for `then` and `or` in which S_1 is completed first and S_2 is only attempted in the callback body with the parameter `satisfied`, which indicates whether the completion of S_1 was successful or not. By contrast, no callback function is invoked in the interpretation of `niob`, and as a result, if there is message passing in S_1, the execution of S_2 will not be blocked as long as it is not dependent on that message passing. It is also notable that in XLCC, binary operators including `then`, `or` and `niob` are not symmetric. If the left-hand-side of `niob` does not involve any message passing, the evaluation will be the same as when the `then` operator is used; that is, the following equivalence holds:

$$S_1 \texttt{ niob } S_2 \Leftrightarrow S_1 \texttt{ then } S_2 \, [if \neg has(S_1, \Leftarrow) \wedge \neg has(S_1, \Rightarrow)] \quad (1)$$

Assume there are n `niobs` appear in a role clause which is split into $n+1$ segments (named as *niob contexts* hereafter). For IMs containing more than one `niob`, we make each context referred to via a specific identifier by which the remaining `defs` can be resumed after the main thread comes back to that context once the awaited message finally arrives.

B. XLCC Semantics

The semantics of XLCC inherits the operational semantics defined in LCC (see in [1]) but XLCC has extended LCC by bringing in new operators and built-in predicates. As mentioned in [1], LCC does not prescribe the means of transmitting messages. However, since XLCC has been designed as a browser-focused choreography script language, the semantics behind message passing needs to be grounded.

1) Messaging: In XLCC, $\Rightarrow$ and $\Leftarrow$ denote sending a message to and receiving a message from another peer respectively. In order to achieve peer-to-peer message passing, any cross-domain messaging protocol could be used here for serving this purpose. By "cross-domain", we mean any messaging client is able to fulfil incoming connections in either a physical manner or a logical manner.

2) Concurrency: XLCC does not employ the `par` operator originally designed in LCC and instead invents the `niob` operator which is inspired by non-blocking I/O to achieve the concurrent computing. `niob` is a binary operator and differentiated from another operator `then` by removing the I/O blocking when the left-hand-side sub-clause is evaluated. As soon as a message passing is encountered, the interpreter will create a callback function and wrap all the remainder of the left-hand-side, which has not been evaluated, into this function. After that, the interpreter will begin to evaluate the right-hand-side sub-clause of `niob` without waiting until the above callback function is fired. If the left-hand-side of `niob` does not involve any message passing, the evaluation will proceed exactly the same as when the `then` operator is applied. Therefore, the `niob` operator can in this case be substituted by the `then` operator, as described in Equation 1.

3) Built-In-Predicates: As of writing this paper, XLCC is still evolving and has the following built-in predicates:

a. `plays` defines which peer will play which role during the IM execution;
b. `knows` defines which OKC(s) the current logged-in peer will provide;
c. `iid` defines the universal ID of an interaction which denotes a one-time execution of a specific IM;
d. `list` replicates the list operations in Prolog, which were part of the original design of LCC.

III. ANNOTATING IMS WITH WSCAIM

In Figure 3, the vocabulary employed inside the IM is unfortunately not machine-interpretable, nor is it easy for humans to interpret; for instance, we can not understand what CC denotes unless the original publisher has added free text comments on this variable. Although the IM publisher could use more self-descriptive labels for arguments, like

$CreditCard$ instead of CC, this does not help unless there is accompanying ontology which provides semantics for the new label, such as http://dbpedia.or-g/resource/Credit_card. IMs without semantic enhancement cannot be properly discovered, understood or repurposed. We propose a framework for semantically enhancing choreography with the *IM Annotator* component as shown in Figure 2. This framework complies with the Linked Data principles [6], thus allowing the annotated choreography to be easily discovered and consumed. We have developed a lightweight choreography ontology named WSCAIM (at http://www.openk.org/wscaim.owl) based on OWL-P [7], the CSP vocabulary (at http://vocab.deri.ie/csp/ and OPENK (originally designed for describing interaction-driven peer communities). Details of our annotation strategy are discussed below where we refine the IM described in Figure 3 by adding two arguments CCC and JTP, which denote the remaining credit in one's credit card and the price for a specific journey option respectively.

A. Process-Dedicated Annotations

Like LCC, XLCC is a process calculus, and consequently we have drawn on OWL-P for terms focused on message passing between peers. OWL-P defines several process-calculus related concepts such as *messages*, *protocols*, *roles*, *propositions* and *commitments*. XLCC uses constraints to restrict peers to their obligations. Therefore, checking if policies inside IMs have been obeyed boils down solving a Constraint Satisfaction Problem (CSP), and the annotations related to this are discussed in Subsection III-B. IMs are annotated and serialised in XHTML+RDFa.

B. Constraint-Dedicated Annotations

We use the CSP vocabulary to annotate the constraint elements of the IM. Figure 4 describes an excerpt of constraint-solving related RDF triples extracted from an IM document and serialised in Turtle. It is notable that the CSP vocabulary does not support comparison between values of variables and without this more expressive annotation, it is difficult if not impossible for IM publishers to annotate constraints on relations between variables. Therefore, we extend CSP with MathML[8] in order to make data comparisons required by IM constraints possible. The triples that realise this extension are also described in Figure 4.

```
@prefix : <http://www.openk.org/ims/JourneyPlanning.html#>.
@prefix csp: <http://vocab.deri.ie/csp#>.
@prefix m3: <http://www.w3.org/1998/Math/MathML/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

:TMCArgumentRelation
 a csp:Relation;
  csp:isSatisfiable "true"^^
   <http://www.w3.org/2011/XMLSchema#Boolean>;
  csp:supports (
   [csp:and
    [csp:var :CC; csp:val :VISA],
    [csp:var :R; csp:val :VISA_REC],
    [m3:apply [m3:geq [m3:ci :CCC], [m3:ci :JTP]]]]
   [csp:and
    [csp:var :CC; csp:val :MASTER],
    [csp:var :R; csp:val :MASTER_REC],
    [m3:apply [m3:geq [m3:ci :CCC], [m3:ci :JTP]]]]).
...
```

Figure 4. Constraint-solving-related triples

IV. Experiment

Our XLCC interpreter's performance on the IM execution with non-blocking I/O is compared with the performance on the I/O-blocking execution in this section. This comparison involved two peers, one of which triggered the interaction by sending messages to the other and received responses later on. Each sending and receiving pair here forms a basic request/response unit (RnR) in which message sending and receiving should happen sequentially. Therefore, the operator `then` was used here to join message sending and message receiving in each RnR and the interaction containing only one unit was not taken into account since I/O has to be blocked by `then` in this case. In the real world, messages passed during peer interactions could be different in length and in order to simplify this, messages lengths were assumed to be equal in this experiment. We experimented with the performance of the XLCC interpreter by calculating the costs of time spent on running IMs with non-blocking I/O (RnRs were joined with `niobs`) and running IMs with blocking I/O (RnRs were joined with `thens`), respectively. After each calculation, the number of units was increased by one. Figure 5 illustrates the result of calculations on the time costs of interactions between two peers based on the above experiment design.

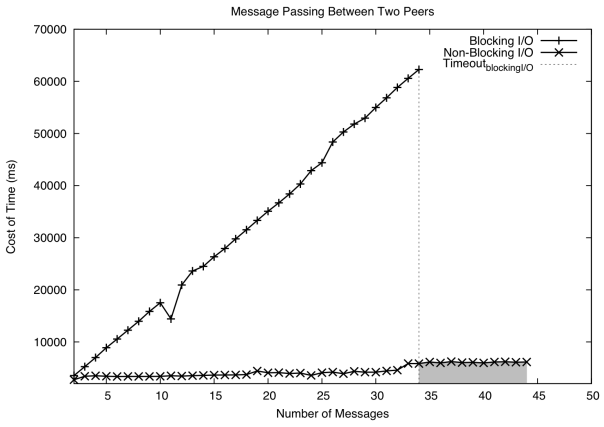

Figure 5. Comparison between peer interactions with non-blocking I/O and blocking I/O

From the above figure, with the increasing of the number of RnR units, there is a steep increase in the time cost of running IM with blocking I/O. However, for the IM executions with non-blocking I/O, the changing of the time cost is relatively trivial and the increase is not obvious.

Moreover, the experiment on the running of IMs with blocking I/O stopped when the number of RnRs reached 35, which occurred due to the timeout of the employed BOSH HTTP endpoint. Needless to say, this could be improved by reconfiguring the BOSH property settings or employing another endpoint with better performance. Nevertheless, with the same timeout configuration, as shown in Figure 5, running IMs in a non-blocking-I/O manner can handle more RnR unites and our framework based on non-blocking I/O therefore scales to the peer-to-peer knowledge sharing environment with a large number of messages being passed around more than the approaches based on the traditional blocking-I/O manner.

V. Related Work

Although several vocabularies inspired by WSDL have been proposed for annotating WS descriptions with semantic markups, most notably OWL-S[9], WSDL-S[10] and SAWSDL[11], little attention has been paid to semantically enhancing the WS choreography which will be launched in the more dynamic environments emerging from open environments and peer-to-peer networks. For example, WS-CDL [12] lacks an appropriate URI-based vocabulary for semantic annotations and other Semantic Web Service solutions such as WSMO [13] are expressive and powerful but also relatively heavyweight, and consequently difficult to apply to portable devices (e.g., mobile phones and tablet PC, etc.). Several lightweight vocabularies have been developed for semantic annotation and targeting at either SOAP-based WSs or RESTful WSs or both, including WSMO-Lite [14], hRESTS [15] (HTML for RESTful Services) and RESTdesc [16]. Also, WS annotation tools have been developed and are still evolving. Crucially, the core concepts for the interaction (or process) specification and markup are still missing in existing choreography approaches of which we are aware. LCC choreography specifications are also executable, but until now LCC interpreters had only been produced for peer-to-peer and service architectures, not for a Web environment. Moreover, LCC still faces concurrency-design issues which have not been tackled in the OpenKnowledge kernel. By contrast, the XLCC script allows publishers to create IMs which can be executed in modern Web browsers.

VI. Conclusions

WS choreography provides a model for representing how peers collaborate with one another in order to achieve their top-level goals. In this paper, we presented OKeilidh as a decentralised proof-of-concept platform which encodes choreography as Interaction Models and executes user agent interactions within modern browsers. XLCC extends the LCC language as a lightweight and browser-focused script language for encoding choreography and its interpreter supports message passing in a peer-to-peer manner. In addition, we have developed a vocabulary that makes it easy for service publishers to annotate services and link them to others in an interconnected manner. This in turn benefits from and enriches the increasing number of resources published in conformity to Linked Data principles.

In future work, we intend to integrate OKeilidh with OKBook [17] (an open online platform for curating peer communities) and thus to make IM publication, IM discovery, IM subscription and IM execution interoperate seamlessly together.

References

[1] D. Robertson, "Multi-agent coordination as distributed logic programming," in *ICLP*. Springer, 2004, pp. 416–430.

[2] B. Suda, *Using Microformats*. O'Reilly Press, 2006.

[3] B. Adida, M. Birbeck, S. McCarron, and S. Pemberton, "RDFa in XHTML: Syntax and processing, W3C Recommendation," http://www.w3.org/TR/rdfa-syntax/, 2008.

[4] I. Hickson, "HTML Microdata," http://www.w3.org/TR/microdata/, 2011.

[5] A. Barker, P. Besana, D. Robertson, and J. B. Weissman, "The benefits of service choreography for data-intensive computing," in *CLADE at HPDC*. ACM, 2009, pp. 1–10.

[6] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data - The Story So Far," *International Journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22, 2009.

[7] A. U. Mallya, N. Desai, A. K. Chopra, and M. P. Singh, "OWL-P: OWL for protocol and processes," in *AAMAS*. ACM, 2005, pp. 139–140.

[8] D. Carlisle, P. Ion, and R. Miner, "Mathematical markup language (MathML), W3C Recommendation," http://www.w3.org/TR/MathML3/, 2010.

[9] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic markup for Web Services. W3C Member Submission," http://www.w3.org/Submission/OWL-S/, 2004.

[10] R. Akkiraju, "Web Service semantics-WSDL-S," http://www.w3.org/Submission/WSDL-S/, 2005.

[11] J. Farrell and H. Lausen, "Semantic annotations for WSDL and XML schema, W3C Recommendation," http://www.w3.org/TR/2007/REC-sawsdl-20070828/, 2007.

[12] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon, "Web services choreography description language version 1.0, W3C Working Draft," http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/, 2004.

[13] R. Lara, D. Roman, A. Polleres, and D. Fensel, "A conceptual comparison of WSMO and OWL-S," in *ECOWS*. Springer, 2004, pp. 254–269.

[14] T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel, "WSMO-lite annotations for Web Services," in *ESWC*. Springer, 2008, pp. 674–689.

[15] J. Kopecký, K. Gomadam, and T. Vitvar, "hRESTS: An HTML Microformat for describing RESTful Web Services," in *WI-IAT*. IEEE CS, 2008, pp. 619–625.

[16] R. Verborgh, T. Steiner, D. Deursen, R. Van de Walle, and J. Valles, "Efficient runtime service discovery and consumption with hyperlinked RESTdesc," in *NWeSP*. IEEE CS, 2011, pp. 373–379.

[17] X. Bai, W. Vasconcelos, and D. Robertson, "OKBook: Peer-to-peer community formation," in *ESWC*. Springer, 2010, pp. 106–120.